This is CS50

Week 3
Today

• How can we compare algorithms with $O$ and $\Omega$ notation?
• What are structs?
• How can we make use of recursion?
Searching and Sorting

(and $O$ and $\Omega$ notation)
Matthew
Samia
Cecelia
How many steps did each algorithm take?

| Linear Search | Binary Search |
How many steps did each algorithm take?

<table>
<thead>
<tr>
<th>Linear Search</th>
<th>Binary Search</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>
What’s the greatest number of steps this algorithm will ever take?

<table>
<thead>
<tr>
<th>Linear Search</th>
<th>Binary Search</th>
</tr>
</thead>
</table>

What's the greatest number of steps this algorithm will ever take?

<table>
<thead>
<tr>
<th>Linear Search</th>
<th>Binary Search</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>( \log_2(7) )</td>
</tr>
</tbody>
</table>
What's the greatest number of steps this algorithm will ever take?

<table>
<thead>
<tr>
<th>Linear Search</th>
<th>Binary Search</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N$</td>
<td>$\log_2(N)$</td>
</tr>
</tbody>
</table>
What’s (approximately!) the greatest number of steps this algorithm will ever take?

<table>
<thead>
<tr>
<th>Linear Search</th>
<th>Binary Search</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O(N)$</td>
<td>$O(\log(N))$</td>
</tr>
</tbody>
</table>
Linear Search
Linear Search
Binary Search
Binary Search

Aaron  Amulya  Alex  Alyssa  Cecelia  Lucas  Ramya
How many steps did each algorithm take?

<table>
<thead>
<tr>
<th>Linear Search</th>
<th>Binary Search</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
How many steps did each algorithm take?

<table>
<thead>
<tr>
<th>Linear Search</th>
<th>Binary Search</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
What's the *fewest* number of steps this algorithm could ever take?

| Linear Search | Binary Search |
What's the *fewest* number of steps this algorithm could ever take?

<table>
<thead>
<tr>
<th>Linear Search</th>
<th>Binary Search</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
What’s (approximately!) the fewest number of steps this algorithm will ever take?

<table>
<thead>
<tr>
<th>Linear Search</th>
<th>Binary Search</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \Omega(1) )</td>
<td>( \Omega(1) )</td>
</tr>
</tbody>
</table>
Thought Question

• Suppose that you create a new algorithm and assess its runtime.
• The fewest steps this algorithm will ever take is 2, and only 2.
• What is the $\Omega$ notation for this algorithm?
Common Notations

- $O(1)$
- $O(\log(N))$
- $O(N)$
- $O(N^2)$

- $\Omega(1)$
- $\Omega(\log(N))$
- $\Omega(N)$
- $\Omega(N^2)$
Sort
<table>
<thead>
<tr>
<th>Algorithm</th>
<th>$O$</th>
<th>$\Omega$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Merge Sort</td>
<td>$O(N\log(N))$</td>
<td>$\Omega(N\log(N))$</td>
</tr>
<tr>
<td>Selection Sort</td>
<td>$O(N^2)$</td>
<td>$\Omega(N^2)$</td>
</tr>
<tr>
<td>Bubble Sort</td>
<td>$O(N^2)$</td>
<td>$\Omega(N)$</td>
</tr>
<tr>
<td>Algorithm</td>
<td>reversed50000.txt</td>
<td>sorted50000.txt</td>
</tr>
<tr>
<td>-------------</td>
<td>------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>Sort1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sort2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sort3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Structs
typedef struct {
   string name;
   int votes;
} candidate;
typedef struct {
    string name;
    int votes;
} candidate;

Create a new "type", which holds a collection of other basic types.
typedef struct {
    string name;
    int votes;
} candidate;

Give the struct a name that can be re-used in the rest of the file.
typedef struct
{
    string name;
    int votes;
}
candidate;

Known as a structure’s members.
typedef struct {
    string name;
    int votes;
} candidate;

candidate president;
typedef struct
{
    string name;
    int votes;
}
candidate;

candidate president;
president.name = "Samia";
typedef struct {
    string name;
    int votes;
} candidate;

candidate president;
president.name = "Samia";
president.votes = 10;
typedef struct
{
    string name;
    int votes;
}
candidate;

candidate candidates[4];
Most Votes

• Create an array of candidates.

• Search the array to find the most votes awarded to any single candidate.

• Print out that candidate’s name.
Recursion
Factorial

1! = 1

2! = 2 * 1

3! = 3 * 2 * 1

4! = 4 * 3 * 2 * 1
Factorial

1! = 1

2! = 2 \times 1

3! = 3 \times 2 \times 1

4! = 4 \times 3 \times 2 \times 1
Factorial

4! = ?
Factorial

4! = 4 * 3!

Recursive call
Factorial

4! = 4 \times 3!
3! = 3 \times 2!
2! = 2 \times 1!
1! = 1

Base case
Factorial

4! = 4 * 3!
3! = 3 * 2!
2! = 2 * 1!
1! = 1

Call stack
Factorial

\[ 4! = 4 \times 3! \]
\[ 3! = 3 \times 2! \]
\[ 2! = 2 \times 1 \]
Factorial

4! = 4 * 3!
3! = 3 * 2 * 1
Factorial

4! = 4 * 3 * 2 * 1
Creating a Factorial Function

• In a file called `factorial.c`, implement a function called `factorial` to return the factorial of a given number.

• Call `factorial` from `main` and print the result from `factorial`. 
This is CS50

Week 3